Choreography for Marionettes: Imitation, Planning, and Control

Todd D. Murphey
Electrical and Computer Engineering
University of Colorado at Boulder
Boulder, Colorado 80309
murphey@colorado.edu

Magnus Egerstedt
Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
magnus@ece.gatech.edu

***Abstract*— This paper presents a project aimed at the creation of fully automated marionettes for puppet plays, to be used in stage plays at the Atlanta Center for Puppetry Arts. In fact, marionettes are sophisticated and challenging mechanical systems, and therefore represent good test-beds for many current issues in robotics, such as systematic modeling of relatively high degree of freedom systems, semantics of high-level motion planning and control, and numerical optimization techniques for motion imitation and generation. This paper presents our approach to these problems, and highlights how insights from puppeteers can aid in the creation of a systematic framework for modeling and control of these complex mechanical systems.**

I. Introduction

A. Choreography as Abstraction

Marionettes are sophisticated mechanical systems, often having 40-50 degrees of freedom, with highly nonlinear constrained dynamics and degenerate Lagrangians (due to the strings having nearly no mass). Moreover, there are typically several of them on the stage simultaneously, often interacting in intricate ways. Despite these nominal obstructions to planning, control, and coordination, we know from watching puppeteers that marionettes may be controlled to act out complex plays as a form of storytelling and performance. To achieve this, puppeteers use standardized choreographic languages as well as systematic approaches to motion imitation [2], [8].

In this paper, we discuss how it is possible to translate the techniques and approaches employed by puppeteers into an automatic framework, capable of mapping high-level choreography into motion commands for the motors that control the strings that suspend the puppets. This way of translating high-level specifications to low-level control commands has applications beyond puppeteering. Examples in which this approach has proved useful include assembly of multi-agent systems [17], task specifications for mobile robots [3], [22], and mission-level control of UAVs [18].

As the puppetry plays as well as the marionettes themselves are highly complex, a hierarchy of interrelated tasks is required to choreograph a mechanized play, seen in Fig.1. Each level of this hierarchy depends in some way or another on the levels below it, and sometimes on the levels above it. Fortunately, if someone is describing how they might enact a play, they naturally include many different levels of abstraction in their description. In fact, as discussed shortly, professional puppeteers explicitly discuss abstraction when describing how they animate the puppets.

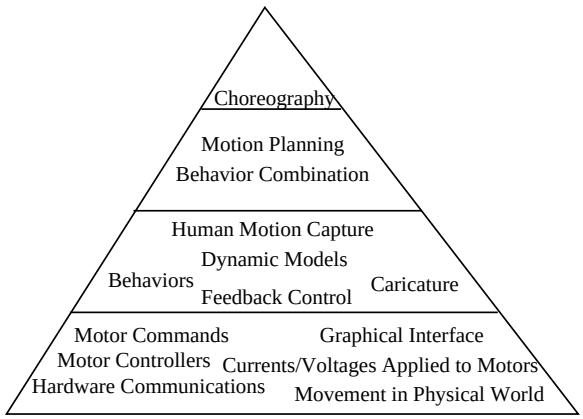


Fig. 1. A hierarchy of capabilities needed to create a marionette play

The use of abstractions allows a puppetry story to be "programmed" in terms of discrete components, tied together by underlying controllers. This involves the construction of a high-level motion description language that can then be "compiled" (traditionally cognitively by a human puppeteer, but in this case by a computer) into actual motion commands. As an example, consider a "play" involving a marionette that needs to switch between walking, hopping, dancing, and jumping. The idea is that this play should be implemented at a high level as a sequence of tokens, each containing the description of an individual motion (possibly together with additional information, such as the duration and location of each motion.) The lower-level control code is then obtained by "compiling" this sequence of tokens into a sequence of control laws that are used for actually producing the desired effect. The futility of doing this without abstraction is evident. Assuming that each puppet is controlled with a minimum of eight actuators, and that most puppet plays have several puppets acting at any given time, hand tailoring every movement is not a realistic solution to creating a play. Instead, one needs a library of abstracted motions that one can choose among when creating the play.

An ideal marionette platform is seen in Fig.2(a), while our current implementations are seen in Fig.2(b) and Fig.2(c). These platforms consist of independent computers to control each marionette, each with an FPGA or microprocessor controlling the motors. Hence, each marionette already have many embedded processors all communicating with each other. In order to implement choreography, each computer

will typically need to have a distribution of the choreography of the play, and many behaviors will require synchronization of all the processors running concurrently. As such, the need for concurrency and questions involving synchronization, parallelization, and embedded control design, are natural issues in this setting. Fortunately, as we will discuss shortly, traditional puppet choreography already systematically deals with many of these basic issues.

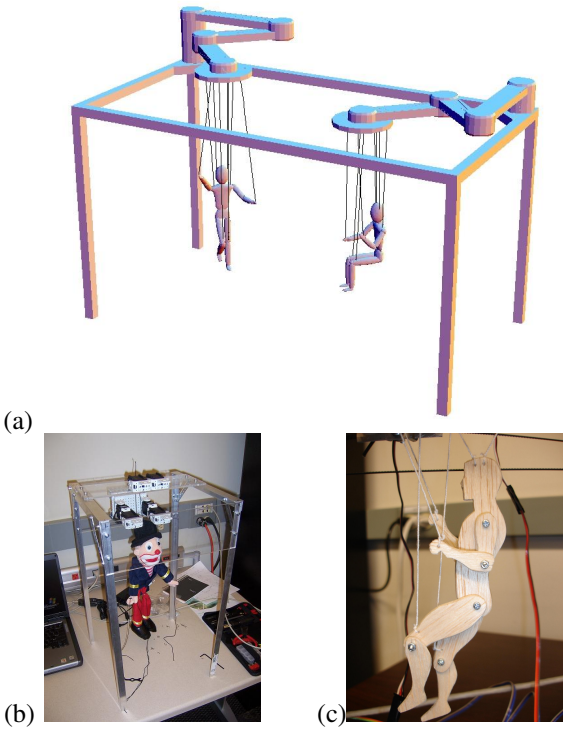


Fig. 2. (a) A *Mathematica* graphical representation of the mechanical system we plan to eventually build; (b) a first-generation 3D experimental puppet platform at the Georgia Institute of Technology that cannot translate in the environment; (c) a first-generation 2D experimental puppet platform at the University of Colorado that can translate in its environment.

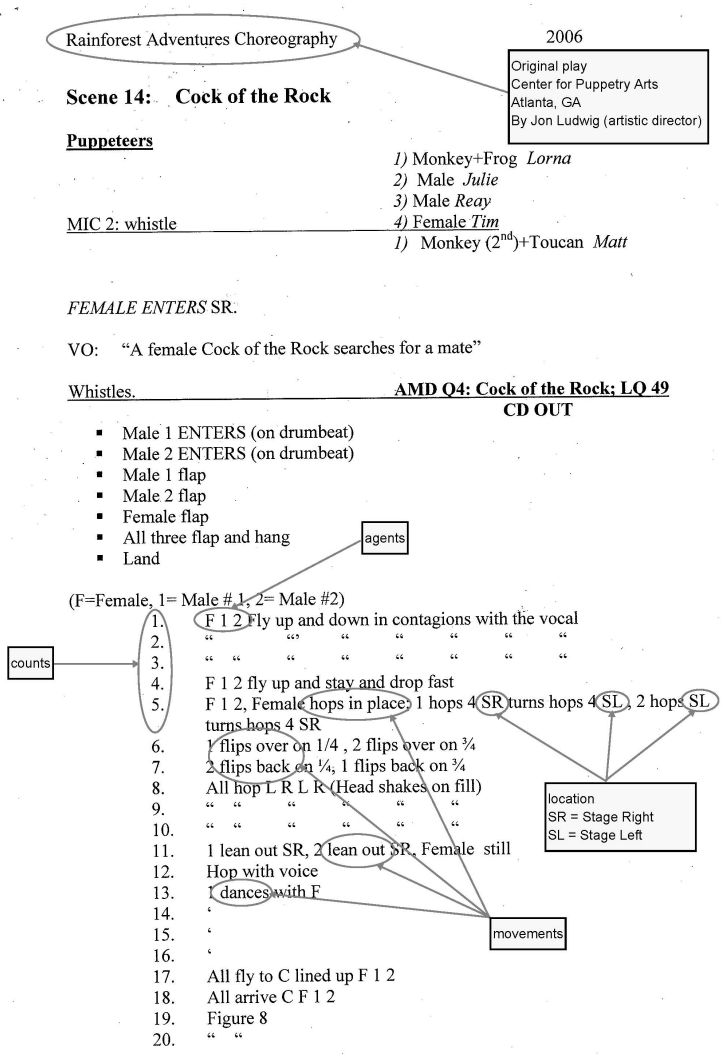


Fig. 3. Rainforest Adventures: This figure is an original puppet choreography sheet from the Center for Puppetry Arts in Atlanta. It shows how the basic building blocks for a formal language for puppet choreography can be derived from existing practices in puppeteering.

B. Motion and Caricature: "Imitate, Simplify, Exaggerate"

The work presented in this paper is based on a collaboration between the authors and professional puppeteers at the Atlanta Center for Puppetry Arts. In order to be able to tap into the experience of the puppeteers, original scores and screenplays have been obtained and analyzed in order to arrive at a language for puppet plays that is expressive enough to capture what the puppets should be doing. In fact, the standard way in which puppet plays are described is through four parameters, namely temporal duration, agent, space, and motion (when?, who?, where?, and what?) [2], [8] (called the *choreographic arguments*). Most plays are based on "counts" in that each puppet motion is supposed to happen at a particular count. (This becomes even more important if multiple puppets are acting simultaneously on stage or if the play is set to music). At each specified count, a motion is initiated and/or terminated. Moreover, the stage is divided into discrete regions. Introducing this explicit granularity into the choreography helps simplify the semantics of the choreography and synchronize motion. In addition, it helps with conflict resolution, such as the deadlock that can occur if two puppets are supposed to be in the same place at the same time. Following these standard practices in the traditional puppet choreography, we, in this paper, propose a formal language for describing puppet plays.

Professional puppeteers use an expression, "Imitate, Simplify, Exaggerate," to describe the basic steps in making a puppet perform a given behavior [8]. First, imitate a behavior that one observes, then simplify it down to its basic components, and finally exaggerate the resulting behavior to convey the same basic level of action and emotional content to the viewer, who is often quite distant from the stage. (In Figure 3, an original puppet choreography example is shown; this is the play *Rainforest Adventures*, by Jon Ludwig, Artistic Director at the Atlanta Center for Puppetry

Arts [9], and the play has been performed throughout the Spring of 2006 in Atlanta [21].) These three stages have formal mathematical counterparts, as discussed Section III. All these stages require simulation, however, discussed next in Section II.

II. Simulation and Bisimulation

We have already developed fast, stable simulators for systems articulated by cables [15], based on variational integrators [23], [14]. In this setting, the Lagrangian $L(q, \dot{q})$ defines an action $A = \int_{t_0}^{t_f} L(q, \dot{q})dt$ that can be used to derive the classical Euler-Lagrange equations. However, it can also be discretized into a sum of integrals $A = \sum_{i=i_0}^{i_f} \int_{i\Delta t}^{i\Delta t+\Delta t} L(q, \dot{q})\mathrm{d}t$. Approximating the action integral provides a direct means of computing the state evolution of the system [23]. Moreover, numerical methods based on this technique preserve energy and momentum characteristics. We have used this approach to create faster-than-realtime simulations that are physically accurate.

Despite the fact that we have a system that does not admit ordinary differential equation analysis because of the numerical calculations being prohibitively slow (and sensitive to numerical error), we can simulate the system in a context where the configuration space is kept small. This is in contrast to standard techniques in fast animation where the configuration space is typically $SE(3)^m$ for m rigid bodies in the system. This small configuration space is what makes optimal control possible.

Euler-Lagrange simulations are typically carried out by symbolically deriving the equations of motion and numerically integrating the resulting differential equations. In [14], we represent systems such as the marionette as an algebraic graph G and then numerically calculate the equations of motion directly based on elements of the graph. Although we do not go into the details of how this construction works here, this specification of complex mechanical systems allows reasonably fast evaluation of the equations of motion while keeping them in the generalized coordinates. We discuss the two settings in which we compute numerical simulation next.

A. Continuous Lagrangian Dynamics

The Euler-Lagrange equations are convenient for control theory applications, particularly for complex mechanical systems where a Newton-Euler approach is difficult. Given a mechanical system with configuration q, the dynamics are described by the Euler-Lagrange equation, where the state space should as small as possible for purposes of determining controllability, observability, reducibility, and other control-related analysis.

$$\frac{\partial}{\partial t}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = \vec{u}(q, \dot{q}, t) \tag{1}$$

where $\vec{u}(q, \dot{q}, t)$ are generalized forces applied to the system expressed in the configuration coordinates. We can expand this equation:

$$\frac{\partial L}{\partial \dot{q}\partial \dot{q}}\ddot{q} + \frac{\partial L}{\partial q\partial \dot{q}}\dot{q} - \frac{\partial L}{\partial q} = \vec{u}(q, \dot{q}, t) \tag{2}$$

If the operator $\frac{\partial L}{\partial \dot{q}\partial \dot{q}}$ is invertable, (2) can be solved to find $\ddot{q}$:

$$\ddot{q} = \left(\frac{\partial L}{\partial \dot{q}\partial \dot{q}}\right)^{-1}\left(\vec{u}(q, \dot{q}, t) + \frac{\partial L}{\partial q} - \frac{\partial L}{\partial q\partial \dot{q}}\dot{q}\right) \tag{3}$$

If all the terms in (3) can be computed numerically at every time step, we can numerically integrate the equation to simulate the system over a period of time. Using the graph representation, we can compute (3) numerically [14]. Assuming that we can numerically evaluate $\vec{u}(q, \dot{q}, t)$, these equations allow us to calculate $\ddot{q}$ in (3) given t, q, and dq without having to symbolically derive the equations of motion. However, ordinary differential equation (ODE) representations of a system can have bad energy characteristics and moreover do not handle impacts and other forms of contact in a transparent manner. Hence, other methods of integrating the equations of motion are useful, as discussed next.

B. Discrete Lagrangian Dynamics

There has recently been a great deal of research in novel methods of numeric integration for mechanical systems that do not depend on ODEs. A result of this research is a class of integrators called variational integrators. It has been proven that these integrators can perfectly conserve various symmetries of mechanical systems, such as momentum, total energy, and the symplectic form [20]. In discrete mechanics, we find a sequence $\{(t_0, q_0), (t_1, q_1), \ldots, (t_n, q_n)\}$ that approximates the actual trajectory of a mechanical system ($q_k \approx q(t_k)$). For simplicity, we assume a constant timestep ($t_{k+1} - t_k = \Delta t \ \forall \ k$), but in general, the timestep can be varied to use adaptive timestepping algorithms.

The variational integrator is derived by defining the discrete Lagrangian which approximates the action integral over a short interval.

$$L_d\left(q_k, q_{k+1}\right) \approx \int_{t_k}^{t_{k+1}} L(q(\tau), \dot{q}(\tau))\mathrm{d}\tau \tag{4}$$

Using the discrete Lagrangian, the system's action integral is replaced with an action sum.

$$\begin{aligned} S(q([t_0, t_f]) &= \int_{t_0}^{t_f} L(q(\tau), \dot{q}(\tau))\mathrm{d}\tau \\ &\approx \sum_{k=0}^{n-1} L_d\left(q_k, q_{k+1}\right) \end{aligned} \tag{5}$$

Minimizing (5) with a discrete variational principle leads to an implicit difference equation known as the discrete Euler-Lagrange (DEL) equation[1],

$$D_1 L_d\left(q_k, q_{k+1}\right) + D_2 L_d\left(q_{k-1}, q_k\right) = 0 \tag{6}$$

an analog of Eq. (1). A root-finding algorithm solves (6) to find q_{k+1} and the process is iterated.

[1] $D_n f(\ldots)$ is the derivative of $f(\ldots)$ with respect to its n-th argument. This is sometimes called a *slot derivative*

As an example, we consider a simple variational integrator using a generalized midpoint approximate for the discrete Lagrangian.

$$L_d(q_k, q_{k+1}) = L\left(q_k + \alpha\left(q_{k+1} - q_k\right), \frac{q_{k+1} - q_k}{\Delta t}\right)\Delta t \tag{7}$$

where $\alpha \in [0,1]$ is a constant and $\alpha = \frac{1}{2}$ leads to second order accuracy [23]. A root-finding algorithm solves (6) to find the next state and the process is then iterated. Forces and constraints can also be represented using the graph representation, just as they can be in the ODE setting. This allows us to simulate an underactuated, constrained system such as a marionette.

C. Bisimulation using kinematic reduction on graphs

Using lower dimensional characterizations of a physical system that preserve the trajectory information (i.e., bisimulations [3], [26], [27]) is appealing for several reasons. First, it can reduce the computational complexity of calculations by reducing the dimension of the state. Second, it can help by abstracting certain components of a complex system, which can be convenient for modeling and implementation purposes. (For instance, a stepper motor is typically abstracted as a first-order linear system in order to avoid the complexities of the underlying nonsmooth phenomena in its construction.) Third, it can improve robustness by using low-level controllers to enforce the abstraction[25].

We used all three advantages in [15], where a mixed kinematic-dynamic description of a marionette was used for purpose of motion planning and simulation. The marionette was assumed to be a dynamic subsystem while the mechanical platform controlling it was assumed to be kinematic. This allows us to simulate the marionette without having to assume a particular dynamic structure of the platform. Moreover, since we are using motors with embeddded controllers, abstracting their control to velocity inputs is feasible.

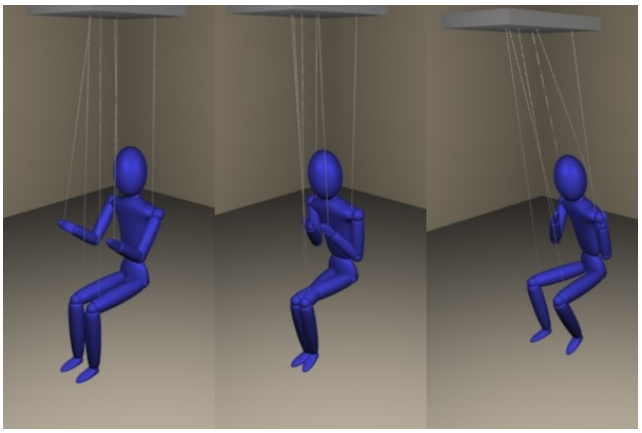


Fig. 4. The tree representation can handle the complex dynamics of a marionette. For a movie of this example, see *http://puppeteer.colorado.edu*.

The key to generalizing this bisimulation strategy is that the calculations must be done directly on the graph G that describes the system, much as the simulations themselves are being calculated directly on the graph. We know that kinematic reducibility depends on the covariant derivative ∇, which can be used to describe the mechanics of simple mechanical systems [4] using $\tilde{\nabla}_{\dot{q}}\dot{q} = u_i Y^i$ where $\tilde{\nabla}$ is the constrained affine connection, u^i are the m inputs, and Y_i are the associated input vector fields. In the unconstrained case, it is defined by $\nabla_X Y = \left(\frac{\partial Y^i}{\partial q^j}X^j + \Gamma^i_{jk}X^j Y^k\right)\frac{\partial}{\partial q^i}$, where the *Christoffel symbols* Γ^i_{jk} associated with the metric g are $\Gamma^i_{jk} = \frac{1}{2}g^{il}\left(\frac{\partial g_{jl}}{\partial q^k} + \frac{\partial g_{kl}}{\partial q^j} - \frac{\partial g_{jk}}{\partial q^l}\right)$ where summation over repeated indices is implied, and upper indices indicate the inverse. which amounts to taking partial derivatives of the vector fields and the kinetic energy relative to the state. From the work in Section II, we know that these quantities can be computed directly based on the graph G that describes the system. Kinematic reduction is then calculated by testing the following condition: $\langle Y_i, Y_j\rangle \in span\{Y_k | 1 \le k \le m\}$ (where $\langle X, Y\rangle = \tilde{\nabla}_X Y + \tilde{\nabla}_Y X$), yielding a reduction to $\dot{q} = \tilde{u}^i X_i$. Since this calculation only depends on calculation of partial derivatives, it can be distributed across the graph in much the same manner that the Euler-Lagrange Equations and variational integrators are computed. Other types of analytical tools that use the covariant derivative, such as controllability and nonlinear stability analysis [4], can also be computed in this setting. Moreover, subgraphs of G can be analyzed without modifying the procedure. We are still in the process of showing that systems with constraints (i.e., using $\tilde{\nabla}$ instead of ∇) can be analyzed in this manner.

III. A Formal Language for Individual Puppet Motions

A system with limited expressive powers, such as a marionette, needs to be able to convey the proper emotions in such a way that a human audience understands what is being conveyed. As previously mentioned, puppeteers achieve this through the three steps: Imitate, Simplify, and Exaggerate. We make these three steps formal in that human motion is being mimicked, after which the resulting motions are projected onto the constrained space over which the marionette operates, followed by a transformation of the resulting motion in such a way that the "energy" of the original (non-projected) motion is reproduced.

Assume that the play starts at time t_0 and that it ends at time t_f. Moreover, let the temporal resolution (the length of each "count") be Δ, and assume that $(t_f - t_0)/\Delta = M$. Following this, the set of all times over which the play is specified is $\mathcal{T} = \{t_0, t_0 + \Delta, t_2 + 2\Delta, \ldots, t_0 + M\Delta\}$.

Moreover, assume that the stage is divided into N different sections (typically this number is 6, namely LowerLeft, LowerCenter, LowerRight, MiddleLeft, MiddleCenter, MiddleRight, UpperLeft, UpperCenter, UpperRight), whose planar center-of-geometry coordinates are given by $r_1, \ldots, r_N$, with the set of regions being given by $\mathcal{R} = \{r_1, \ldots, r_N\}$.

Assume that each puppet has dynamics given in standard form [16] by $\dot{x}^i = f^i(x^i, u^i)$, $y^i = h^i(x^i)$, where the superscript i denotes agent i. Note that we can obtain f^i directly from the graph-based methods described in Sec-

tion II. Moreover, the state $x^i \in X^i \subset \Re^{n^i}$, and the output $y^i \in \Re^m$ is given by a mapping h^i from X^i to a lower dimensional space $\Re^m$. Now, supposing that we have constructed a number of control laws $\kappa^i_j,\ j = 1 \ldots, C^i$, corresponding to different motions that puppet i can perform, with each control law being a function of x^i (state), t (time), and α_i (a parameter characterizing certain aspects of the motion such as speed, energy, or acceleration, as is the normal interpretation of the parameterization of biological motor programs [5], [12]), we can let the set of moves that puppet i can perform be given by $\mathcal{K}^i = \{\kappa^i_1, \ldots, \kappa^i_{C^i}\}$. In fact, we will often use the shorthand $f^i_j(x^i, t, \alpha)$ to denote the impact that control law κ^i_j has on puppet i through $f^i(x^i, \kappa^i_j(x^i, t, \alpha))$.

As already pointed out, each instruction in the puppet play language is a four-tuple designating *when, who, where*, and *what* the puppets should be doing. In other words, given that we have a total of P puppets, with $\mathcal{P} = \{1, \ldots, P\}$, we can let the puppet alphabet associated with puppet i be given by $\mathcal{L}^i = \mathcal{T} \times \mathcal{T} \times \mathcal{R} \times \mathcal{K}^i$. Each element in $\mathcal{L}^i$ is thus given by (T_0, T_1, r, κ), where the interpretation is that the motion should take place during the time interval $T_1 - T_0$, largely in region r, while executing the control law κ.

And, if we drop the explicit dependence on i, i.e. assume that we are concerned with a given puppet, we can then follow the standard notation in the formal language field [5], [11] and let $\mathcal{L}^\star$ denote the set of all finite-length concatenations of elements in $\mathcal{L}$ (including the empty string), and let puppet plays be given by words $\ell \in \mathcal{L}^\star$. In particular, if we let $\ell = (t_0, T_1, r_1, \kappa_1), \ldots, (T_{p-1}, T_p, r_p, \kappa_p)$ be p motions in a row, then the puppet operates on this string through

$$\dot{x} = \begin{cases} f_1(x, t, \alpha_1),\ t \in [t_0, T_1) \\ f_2(x, t, \alpha_2),\ t \in [T_1, T_2) \\ \quad \vdots \\ f_p(x, t, \alpha_p),\ t \in [T_{p-1}, T_p]. \end{cases}$$

We thus have obtained the semantics of a motion description language that allows us to specify 'plays' at a high-level of abstraction through strings over a motion alphabet. However, this construction only becomes meaningful if we can populate the underlying motion alphabet with appropriate motion primitives, which is the topic of the next section.

A. Constructing Components of the Language

The approach just described seems fairly natural, but two essential parameters seem to have been left out. First, the parameterized motion parameters $\alpha_1, \ldots, \alpha_p$ have not yet been specified. Moreover, the desired regions $r_1, \ldots, r_p$ have not been utilized in any way. In order to remedy this, we need to construct not just a dynamic "parser" for puppet plays, as given above, but also a "compiler" that selects the "best" parameters (as well as durations) for the different moves so that the play is executed as efficiently as possible.

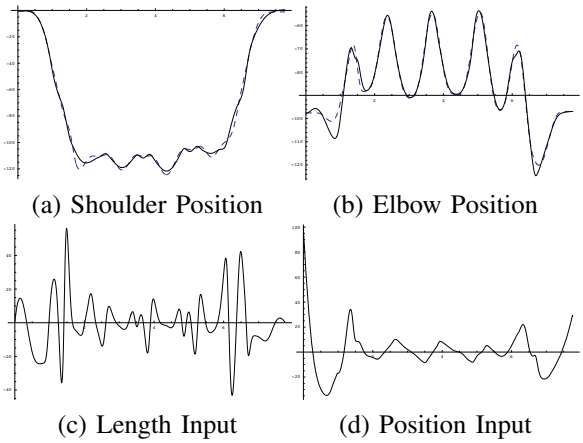
However, before one can embark on the task of selecting the best member in a parameterized class of control laws, this class must be obtained in the first place. If the puppet is a human marionette, we would like it to execute human-like motions. We have been using data from Carnegie Mellon University [19]. However, since marionettes are constrained in such a way that they cannot be as expressive as human motions, we first identify human motions (corresponding to the Imitate phase in puppetry), project human motion down onto the space of available puppet motions (the Simplify phase), and then exaggerate these motions in order to make them sufficiently expressive (the Exaggerate phase). Formally speaking, given a desired trajectory $z(t) \in Z$ that we would like the puppet (whose state is $x(t) \in X,\ \dim(X) \leq \dim(Z)$) to follow, we define a projection $\pi : Z \to X$. This is typically simply the immersion defined by the subgraph of the marionette graph G_M that is isomorphic to the human graph G_H. The Simplify-phase thus consists of minimizing expressions like accumulated error,

$$\int_{t_0+i\Delta}^{t_0+(i+1)\Delta} L(x(t) - \pi(z(t)))dt, \tag{8}$$

(where $L = \|x(t) - \pi(z(t))\|^2$) subject to the puppet dynamics $\dot{x} = f(x, u)$.We use a projection-operator based approach to trajectory optimization [10]. It searches the trajectory space of the target system for an admissible trajectory that best resembles the desired one. The returned trajectory is always admissible, even if the system dynamics are nonlinear, uncontrollable, or unstable.

To implement this optimization procedure, we use a nonlinear projection operator $\mathcal{P}$ to project trajectories onto the manifold of admissible trajectories $\mathcal{T}$. This projection depends on calculating the linearization of the system equations, which we can do using the graph representation discussed in Section II. The projection operator approach uses a novel modification of standard constrained optimization techniques. Instead of a constrained search in the admissible trajectory space of the system, it performs an unconstrained search in the larger general trajectory space and uses a projection operator to project trajectories into admissible trajectories. In this setting, the search is roughly analogous to an optimization in finite dimensions using an iterative gradient descent method. At each iteration, a new descent direction is found by solving a linear optimal control problem. An Armijo search is performed in the descent direction until it finds a new trajectory that improves the current score. The new trajectory is projected back into the admissible trajectory space, and the process is repeated. Details of this analysis may be found in [13].

An example of such a calculation is seen in Fig. 5, where a simplified string puppet is given a waving motion and then an optimal control procedure computes string motions that imitate the waving motion. Fig. 5(a) and (b) show the shoulder and elbow trajectories. The dashed line is the desired trajectory that we might see from a human waving, while the solid line represents the trajectory that results from the optimization procedure. It is important to note that these two trajectories match at nearly all times, indicating that the problem can be solved. However, the inputs required to solve



(a) Shoulder Position (b) Elbow Position

(c) Length Input (d) Position Input

Fig. 5. Imitation: A 2-dimensional puppet arm waving. Plots (a) and (b) show the desired (dashed) and obtained (solid) angles of the shoulder and elbow based on motion-capture data, while plots (c) and (d) show the optimized string length and position inputs. A movie of the optimized motion may be found at *http://puppeteer.colorado.edu*

the problem (seen in Fig. 5(c) and (d)) are very complex–one would not want to have to guess them.

B. A Puppet Play Compiler: From Motions to Sequences

Given a series of motions that are dynamically feasible, how do we take strings that correspond to these motions and turn them into actual physical motions? Each motion is described by a start time, a stop time, a region of action, and input. Two such motions in a row can be denoted by $(0, T, r_1, \kappa_1), (T, t_f, r_2, \kappa_2)$.

Ideally, the actual transition time τ between motions will be roughly the same as the desired transitions time T. That is, we'd like to keep $|T - \tau|$ small. Similar goals include: being in the correct region at the transition times; keeping the energy used as small as possible; and ensuring that the reference (e.g., from motion capture) is tracked closely. To frame these goals as an optimal control problem, we simply specify that any deviations from the ideal ones are to be kept as small as possible.

A standard, variational argument from our prior work in [6] gives the optimality conditions for solving this condition. That approach amounts to solving the optimization problem

$$J = \min_{t_i} \sum_{i=0}^{m} \int_{t_i}^{t_{i+1}} L dt. \tag{9}$$

Moreover, we showed that the necessary computations in this setting are feasible for systems as complex as marionettes. By a direct generalization to more than two modes, this construction allows us to produce a compiler that takes sequences of parameterized tokens and outputs sequences of control modes with an optimized temporal duration and mode-parameterization through a gradient descent algorithm (see for example [1], [7]).

An example of this proposed approach is shown in Figure 6, in which an oscillator is switching between two modes - one slow (corresponding to walking) and one fast (corresponding to running).

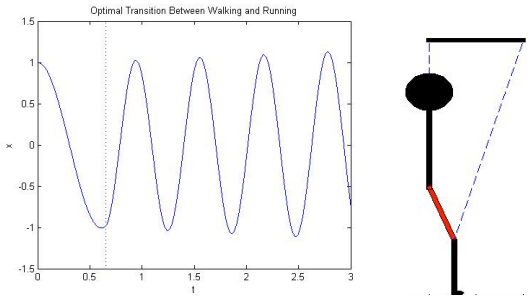


Fig. 6. A simplified locomotion model in which walking and running are defined through linear oscillators with different frequencies. The figure on the left depicts the waveform of the gait, and the figure on the right is a snapshot of the animation showing the result, which can be found at *http://puppeteer.colorado.edu*.

An alternative approach to the one described above is to *first* compute the hybrid modes based on an abstraction (such as a bisimulation strategy like that discussed in Section II), and *then* use a smooth optimization to combine the motions. This entails solving Eq. (9) first for the abstracted system and then solving Eq. (8). If the abstraction has been generated carefully, this can potentially lead to a more numerically efficient algorithm by virtue of being able to *approximate* the solution to Eq. (8). However, if one truly has to solve the full version of Eq. (8), then it will almost certainly be more numerically efficient to solve it offline and solve Eq. (9) online.

IV. Conclusions

We have introduced our approach to motion control and coordination of marionette puppets. Marionettes represent an excellent testbed for testing online embedded system design and control design. Our plan is to continue to use the testbed as a prototype for verification and testing reliability of various control approaches for high-dimensional systems. In the near future we are still implementing each of the pieces described here in the hardware in Fig. 2. Soon, we will use these puppets in a performance with the Atlanta Center for Puppetry Arts. In addition, we believe that the marionette platform is ideally suited for web-based control. Using the simulation method described in Section II, we can implement faster-than-realtime simulation of the marionettes that people can use to simulate their choreography. They will be able to upload it on the project website and see their choreography on a real device over a webcam. This is in the same spirit as the *Demonstrate* (2004) and the *Telegarden* (1995-2004 [24]) projects, where thousands of people used online robotic environments for artistic expression.

V. ACKNOWLEDGMENTS

We would like to thank Jon Ludwig of the Atlanta Center for Puppetry Arts and Annie Peterli, freelance puppeteer extraordinaire.

References

[1] L. Armijo. Minimization of functions having lipschitz continuous first-partial derivatives. *Pacific Journal of Mathematics*, 16:1–3, 1966.

[2] B. Baird. *The Art of the Puppet*. Mcmillan Company, New York, 1965.

[3] C. Belta, V. Isler, and G. J. Pappas. Discrete abstractions for robot planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874, 2005.

[4] F. Bullo and A.D. Lewis. *Geometric Control of Mechanical Systems*. Number 49 in Texts in Applied Mathematics. Springer-Verlag, 2004.

[5] M. Egerstedt and R.W. Brockett. Feedback can reduce the specification complexity of motor programs. *IEEE Transactions on Automatic Control*, 48(2):213–223, February 2003.

[6] M. Egerstedt, T. D. Murphey, and J. Ludwig. *Hybrid Systems: Computation and Control*, volume TBD of *Lecture Notes in Computer Science*, chapter Motion Programs for Puppet Choreography and Control, pages 190–202. Springer-Verlag, 2007. Eds. A. Bemporad, A. Bicchi, and G. C. Buttazzo.

[7] M. Egerstedt, Y. Wardi, and H. Axelsson. Transition-time optimization for switched systems. *IEEE Transactions on Automatic Control*, 51(1):110–115, January 2006.

[8] L. Engler and C. Fijan. *Making Puppets Come Alive*. Taplinger Publishing Company, New York, 1973.

[9] Center for Puppetry Arts. http://www.puppet.org/.

[10] J. Hauser. A projection operator approach to optimization of trajectory functionals. Barcelona, Spain, 2002.

[11] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 2 edition, 2000.

[12] M. Itô. *The Cerebellum and Neural Control*. Raven, New York, 1984.

[13] E. Johnson and T. D. Murphey. Automated trajectory morphing for marionettes using trajectory optimization. In *IEEE Int. Conf. on Robotics and Automation*, 2007. Submitted.

[14] E. Johnson and T. D. Murphey. Discrete and continuous mechanics for tree representations of mechanical systems. In *IEEE Int. Conf. on Robotics and Automation*, 2007. Submitted.

[15] E. Johnson and T. D. Murphey. Dynamic modeling and motion planning for marionettes: Rigid bodies articulated by massless strings. In *IEEE Int. Conf. on Robotics and Automation*, pages 330–335, 2007.

[16] H.K. Khalil. *Nonlinear Systems (second edition)*. Prentice Hall, 1996.

[17] Eric Klavins, Robert Ghrist, and David Lipsky. A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control*, 51(6):949–962, June 2006.

[18] T. J. Koo, J. Liebman, C. Ma, and S. Sastry. Hierarchical approach for design of multi-vehicle multi-modal embedded software. In T. A. Henzinger and C. M. Kirsch, editors, *Embedded Software, Lecture Notes in Computer Science 2211*, pages 344–360. Springer-Verlag, 12001.

[19] J. Lee, J. Chai P. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics*, pages 491 – 500, 2002.

[20] A. Lew, J. E. Marsden, M. Ortiz, and M. West. Asynchronous variational integrators. *Arch. Rational Mech. Anal.*, 167:85–146, 2003.

[21] J. Ludwig. Rainforest adventures. http://www.puppet.org/perform/rainforest.shtml.

[22] V. Manikonda, P.S. Krishnaprasad, and J. Hendler. Languages, behaviors, hybrid architectures and motion control. In Willems and Baillieul, editors, *Mathematical Control Theory*, pages 199–226. Springer-Verlag, 1998.

[23] J. E. Marsden and M. West. Discrete mechanics and variational integrators. *Acta Numerica*, pages 357–514, 2001.

[24] M. L. McLaughlin, K. K. Osborne, , and Nicole B. Ellison. *Virtual Culture*, chapter Virtual Community in a Telepresence Environment, pages 146–168. Sage Publication, London, 1997.

[25] T. D. Murphey. Kinematic reductions for uncertain mechanical contact. *Robotica*, To be published in 2007.

[26] G.J. Pappas, G. Laffierier, and S. Sastry. Hierarchically consistent control sytems. *IEEE Trans. Automatic Control*, 45(6):1144–1160, June 2000.

[27] Paulo Tabuada and George J. Pappas. Bisimilar control affine systems. *Systems and Control Letters*, 52(1):49–58, May 2004.